The Lincoln TX-2 Input-Output System*

JAMES W. FORGIE†

INTRODUCTION

The input-output system of the Lincoln TX-2 computer contains a variety of input-output devices suitable for general research and control applications. The system is designed in such a way that several input-output devices may be operated simultaneously. Since the computer is experimental in nature, and changes in the complement of input-output devices are anticipated, the modular scheme used will facilitate expansion and modification. The experimental nature of the computer also requires that the input-output system provide a maximum of flexibility in operating and programming for its input-output devices.

The input-output devices, currently scheduled for connection to TX-2, include magnetic-tape units for auxiliary storage; photoelectric paper-tape readers for program input; a high-speed printer, cathode-ray-tube displays, and Flexowriters for direct output; analog-to-digital conversion equipment; data links with other computers; and miscellaneous special-purpose equipment. This paper will not be concerned with the details of these devices, but will limit itself to a discussion of the logical incorporation of them into the system.

In describing the TX-2 input-output system, reference will be made to certain design aspects of other parts of the TX-2 as set forth in the previous paper.

THE MULTIPLE-SEQUENCE PROGRAM TECHNIQUE

Of the various organizational schemes which permit the simultaneous operation of many devices, we have chosen the "multiple-sequence program technique" for incorporation in TX-2. A multiple-sequence computer is one that has several program (instruction) counters. If the program sequences associated with these program counters are arranged to time-share the hardware of the central computer, a machine can be obtained which will behave as if it were a number of logically separate computers. We call these logical computers sequences and therefore refer to TX-2 as a multiple-sequence computer. By associating each input-output device with such a sequence, we effectively obtain an input-output computer for each device.

Since the one physical computer in which these sequences operate is capable of performing only one instruction at a time, it is necessary to interleave the sequences if they are to operate simultaneously. This interleaving process can take place aeriodically to suit the needs of and under the control of, whatever individual input-output devices are operating. The number of sequences which can operate simultaneously, and the complexity of the individual sequences, is limited by the peak and average data-handling rate of the central computer hardware.

In a multiple-sequence computer, the main body of the computation can be carried out in any sequence, but if maximum efficiency of input-output operation is to be achieved, the bulk of arithmetic operations must be confined to a few special sequences, called main sequences, which have no associated input-output devices. The input-output sequences may then be kept short, and a large number of them can be executed at once.

MULTIPLE-SEQUENCE OPERATION IN TX-2

In TX-2, one-half of the index-register memory has been made available for storing program counters. Thus, a total of 32 sequences may be operated in the machine. (Actually an additional sequence of special characteristics is obtained by using index register number 0 as a program counter. This special sequence will be discussed later.) Some of these sequences are associated with input-output devices. Others perform functions, such as interpreting arithmetic overflows, that are called into action by conditions arising within the central computer. Finally, there are the main sequences which are intended to carry out the bulk of the arithmetic computations performed by the machine.

A priority scheme is used to determine which sequence will control the computer at a given time. If more than one sequence requires attention at the same time, control of the machine will go to the sequence having the highest priority, and instructions addressed by its program counter will be executed.

Table I is a list of the sequences currently planned for inclusion in TX-2. They are listed in approximate order of priority with the highest at the top. Asterisks mark sequences which are not associated with any particular in-out device. A special sequence (number 0) has first priority and will be used to start any of the other sequences at arbitrary addresses. The next two sequences interpret alarms (under program control). These three sequences have the highest priorities, since they must be capable of interrupting the activities of other sequences. The input-output devices follow, with high-speed, freerunning units carrying next highest priorities. The main sequences (we anticipate three) are at the bottom of the list. The priority of any sequence may be easily changed, but such changes are not under program control. Priorities are intended to remain fixed under normal operating conditions. The list totals about 25 sequences, leaving eight spaces for future expansion.

* The research in this document was supported jointly by the Army, Navy, and Air Force under contract with Mass. Inst. Tech.
† Lincoln Lab., M.I.T., Lexington, Mass.
Switching between sequences is under the control of both the input-output devices (generalized to include alarms, etc.) and the programmed instructions within the sequence.

Once a sequence is selected and its instructions are controlling the computer, further switching is under control of the programmed instructions. Program control of sequence switching is maintained through two bits, called the break and dismiss bits, in each instruction. The break bit governs changes to higher-priority sequences. When the break bit permits a change, and some higher-priority sequence requests attention, a change will be made. The dismiss bit indicates that the sequence has completed its operation (for the moment, at least) and that lower-priority sequences may receive attention. The interpretation of the break and dismiss bits will be discussed in more detail.

**The TX-2 Input-Output Element**

The TX-2 input-output element is shown schematically in Fig. 1. It consists of a number of input-output devices, associated buffers, and a sequence selector. Each device has enough control circuitry to permit it to operate in some selected mode once it has been placed in that mode by signals from the central computer. Associated with each device is a buffer storage of appropriate size. This buffer may be large or small, to suit individual data-rate requirements, but the buffers used in TX-2 will generally be the smallest possible. For the most part, buffering for only one line of data from the device (e.g., 6 bits for a paper-tape reader) will be provided. Each input-output device is associated with one stage of the sequence selector. The sequence selector provides the control information necessary for proper interleaving of the program sequences. When it is desired to add a new input-output device to the computer, the three packages, in-out unit, buffer, and sequence-selector stage, must be provided.

As shown in Fig. 1, data is transferred between the input-output element and the central computer by way of the exchange element. Fig. 1 indicates two-way paths between the E register and all in-out buffers. Actually, most devices are either readers or recorders, but not both, and therefore require one-way paths only. Only the necessary paths are provided; the drawing simply shows the most general case.

Signals from the sequence selector connect the appropriate buffer register to the E register to transfer data. When a sequence is selected (i.e., its program counter is supplying instruction locations), the associated buffer is connected to the E register, and all other buffers are disconnected. A read instruction will effect a transfer of information between the buffer and the E register. A particular buffer is thus accessible only to read instructions in the sequence associated with the buffer’s in-out unit.

Fig. 1 shows paths from the sequence selector to a coder which provides an output called the program-counter number. These paths are used in the process of changing sequences to be described in a later section.

Fig. 1 also shows paths for mode selection in the in-out element. The use of these paths is described in the next section under *ios*.

**Input-Output Instructions**

In addition to the break and dismiss bits on all instructions, the programmer has three computer instructions for operating the input-output system. There are two read instructions, *rdn* and *rds*, which transfer data between the in-out devices and the central computer memory. The third instruction, *ios*, selects the mode of operation of the in-out devices.

*rdn and rds*

Both of the read instructions obtain a word from memory. If the in-out device associated with the sequence in which the read instruction occurs is in a reading (input) mode, appropriate bits of the memory word are altered, and the modified word is replaced in memory. If the in-out device is in a recording (output) mode, appropriate bits of the memory word are fed to the se-
lected in-out buffer, and the word is replaced in memory. Thus, the same \textit{read} instruction suffices for both input and output operations. The distinction between \textit{rdn} and \textit{rd$N$} lies in the assembling of full memory words from short buffer words. An \textit{rdn} instruction will place the 6 bits from a tape reader in the right 6 bits of a 36-bit memory word. The remaining 30 bits will be left unchanged. An \textit{rd$N$} instruction for the same tape reader will place the 6 bits in a splayed pattern (every sixth bit across the memory word) and will shift the entire word one place to the left before replacing it in memory. Except for the shift, the other 30 bits remain unchanged. A sequence of 6 \textit{rd$N$} instructions, one for each of 6 tape lines and all referring to the same memory address, will suffice to assemble a full 36-bit word.

The distinction between \textit{rdn} and \textit{rd$N$} could be obtained from mode information in the in-out device, but the inclusion of both instructions in the order code allows the programmer to interchange the two types freely to suit his needs. The \textit{rdn} instruction makes use of the permutation aspect of the TX-2 configuration control and is, therefore, particularly convenient for dealing with alphanumeric Flexowriter characters. Configuration is not applicable to the \textit{rd$N$} instruction.

\textbf{iios}

The \textit{iios} instruction serves to put a particular in-out device into a desired mode of operation. The \textit{j} bits of the instruction word, normally the index register number, in this case specify the unit number of the in-out device. This number is the same as the program counter number for the associated sequence, although the correspondence is not necessary. The \textit{y} bits of the instruction word specify the mode of operation in which the unit is to be placed. Two of the \textit{y} bits are sent directly to the \textit{jth} sequence selector stage and serve to control the sequence, regardless of the mode of its associated in-out device. These two bits allow \textit{iios} instructions to arbitrarily dismiss or request attention for any sequence in the machine. By means of these instructions, one sequence can start or stop all others in the machine. A third \textit{y} bit determines whether the mode of the in-out device is to change as a result of the instruction. If it is to change, the remaining 15 bits specify the new mode. An \textit{iios} instruction occurring in any sequence can thus start or stop any sequence and/or change the mode of its in-out device.

A further property of the \textit{iios} instruction is that it leaves in the \textit{E} register a map of the state of the specified in-out control prior to any changes resulting from the instruction itself; \textit{iios} instructions may, therefore, be used to sense the state of the in-out system without altering it in any way.

\textbf{SEQUENCE-CHANGING AND OPERATION OF THE SEQUENCE-SELECTOR}

At some point just before the completion of the instruction memory cycle in TX-2, the Control must decide whether the next instruction would be taken from the current sequence or from some new sequence. The information on which this decision must be based comes from the break and dismiss bits of the instruction word currently in use and from the sequence selector. Fig. 2 is a detailed drawing of one stage of the sequence selector. All stages, except that with the highest-priority, are identical. The lowest-priority stage returns the final three control signals to the control element.

Each stage of the sequence selector retains two pieces of information concerning its associated sequence. One flip-flop (ss \textit{j}.1) remembers whether or not the sequence is selected (\textit{i.e.}, whether or not it is receiving attention). The priority signal (labeled \textit{no higher priority sequence requests attention}) passes from higher to lower priority stages until it encounters a stage which requests, but is not receiving attention. Such a stage is said to have priority at the moment, and its output to the program-counter-number coder prepares the number of the new program counter in anticipation of a sequence change.

The process of changing sequences involves storing the program counter for the old sequence and obtaining the counter for the new. Actually, to speed up the overall process, the new program counter is obtained first, so that it may be used while the old is being stored. Using the paths shown in Fig. 1, the new program counter number is placed in the \textit{j} bits of the \textit{N} register. The new program counter is then obtained from the \textit{X} memory and interchanged with the old program counter contents which have been in the \textit{P} register.\footnote{The \textit{P} register is shown in Fig. 4 of Frankovich and Peterson, this issue, p. 148.} The \textit{K} register, which has been holding the old program counter number since the last sequence change, is now interchanged with the \textit{j} bits, and the old counter is stored at the proper location in the \textit{X} memory. The state of the sequence selector is changed, to conform to the change of sequence, by sending a \textit{select new sequence} command from Control. This command clears the ss \textit{j}.2 flip-flop in the old-sequence stage and sets the ss \textit{j}.2 flip-flop to a \textit{ONE} in the new-sequence stage.\footnote{The relative timing of the central computer actions during the change process is shown in Fig. 6(d) of Frankovich and Peterson, this issue, p. 150.}

\textbf{INTERPRETATION OF THE BREAK BIT}

The programmer uses the \textit{break} bit of an instruction word to indicate whether or not change to a higher priority sequence may occur at the completion of the instruction. The fact that a programmer permits a \textit{break} does not mean that the sequence has completed its current task, but merely that no harm will be done if a change to some higher-priority sequence is made. \textit{Breaks} should be permitted at every opportunity if a number of in-out devices are operating. The sort of situation in which a \textit{break} cannot be permitted occurs when the \textit{E} register is left containing information which the program requires at a later step. If a change occurred in this case, the contents of the \textit{E} register would be destroyed and lost to the program.
When a break is permitted by the current instruction, a sequence change will actually take place only if some higher-priority sequence requests attention. A signal from the sequence selector to the control element provides this information (Fig. 2). When a break type of sequence change is made, the ssj.1 flip-flop in the sequence selector remains unchanged, and the sequence which was abandoned in favor of one of a higher-priority continues to request attention.

**Fig. 2—Block diagram of TX-2 sequence selector stage.**

**INTERPRETATION OF THE DISMISS BIT**

The dismiss bit is used by the programmer to indicate that the sequence presently in use has completed its task. To provide synchronization in the in-out system, dismiss bits must be programmed between attention requests from the in-out devices. In this case, the dismiss operation guarantees that the computer will wait for the next signal from the in-out device before proceeding with the associated program sequence.

The dismiss bit is also used to accomplish the halt function in TX-2. A multiple-sequence computer halts when all sequences have been dismissed and all in-out units turned off. The priority signal from the sequence selector to the control element provides the information as to whether or not any sequence in the machine requests attention. When none request attention, the control stops all activity in the machine as soon as a dismiss bit appears on an instruction in the sequence being used. Activity is resumed in the machine as soon as some in-out device or push button requests attention.

The sequence change which results from a dismiss bit is identical with that resulting from a break except that a dismiss current sequence command accompanies the select new sequence command from Control to the Sequence Selector (Fig. 2).

**STARTING A MULTIPLE-SEQUENCE COMPUTER**

In a single-sequence computer the starting process involves resetting the program counter to some arbitrary value and starting the control. In a multiple-sequence computer, the program counter for a particular sequence must be reset and the sequence started. In TX-2 a special sequence (number 0) has the highest priority and is used to facilitate starting. This sequence has the special feature that its program counter always starts at an initial memory location specified by a set of toggle switches. Attention for the sequence is requested by pushing a button on the console. By executing a short program stored in the toggle-switch registers of the V memory, this sequence can start (or stop) any other sequence in the machine. The starting process for an arbitrary sequence involves resetting its program counter by means of an ids (load index register) instruction, and starting its sequence with an ios instruction.

**THE ARITHMETIC ELEMENT IN MULTIPLE-SEQUENCE OPERATION**

While efficient operation requires that the bulk of arithmetic operations be carried out in a main sequence, the arithmetic element in TX-2 is available to all sequences. Since once a change has been made to a higher-priority sequence, control cannot return to a lower-priority sequence until the higher-priority one has been dismissed, a simple rule allows the arithmetic element to be used in any sequence without confusion. If, whenever a higher-priority sequence requires the arithmetic element, it stores the contents of any registers it will need (A, B, C, D, or F) and reloads them before dismissing, all lower-priority sequences will find the registers as they left them. This storing and loading operation requires time and, therefore, lowers the total data-handling capacity, but the flexibility obtained may well be worth the loss in capacity.

The step-counter class of arithmetic element instructions is a special problem. These instructions can require many microseconds to complete, and while TX-2 is designed to allow in-out and program element instructions to take place while the arithmetic element is busy, the case can arise in which an arithmetic element instruction (load, store, etc.) appears before the AE is finished with a step-counter class instruction. The machine would normally wait in an inactive state until the operation is complete, but since there is a chance that some higher-priority sequence may request attention in the interim and have instructions which can be carried out, provision is made to keep trying changes to higher-priority sequences as they request attention. The machine thus waits in an inactive state only when no higher-priority sequences have instructions which can be performed. This provision allows the programmer to ignore the arithmetic element in considerations of peak- and average-peak rate calculations when he desires to operate a maximum number of in-out devices.

**CONCLUSION**

Multiple-sequence operation of input-output devices, as realized in TX-2, has a number of significant characteristics. Among them are:

1) A number of in-out devices may be operated concurrently with a minimum of buffering storage.
2) Machine time is used efficiently, since no time need be lost waiting for input-output devices to complete their operation. Other machine activity may proceed meanwhile.

3) Each input-output device may be treated separately for programming purposes. Efficiency of operation is obtained automatically when several separately programmed devices are operated simultaneously, although average- and peak-rate limitations must be considered.

4) Maximum flexibility in programming for input-output devices is obtained. The full power of the central machine may be used by each input-output sequence if desired. Routines for each device may be as long or as short as the particular situation requires.

5) The modular organization of the input-output equipment simplifies additions and modifications to the complement of in-out devices.

6) The organization of buffering storage allows the amount and kind of such storage to be tailored to the needs of the individual devices and the data-handling requirements to be met by the system.

7) The multiple-sequence program technique appears to be particularly well suited to the operation of a large number of relatively slow input-output devices of varying characteristics, as opposed to a smaller number of high-speed devices.

Memory Units in the Lincoln TX-2*

RICHARD L. BEST†

Memory Units in the Lincoln TX-2 Computer

There are 3 high-speed live memories in TX-2; all are random access and all use ferrite cores. The largest is the 6½-μsec cycle time “S” memory with 65,536 37-digit words. The “T” memory is entirely transistor driven; it has a capacity of 4096 37-digit words and a 5½-μsec cycle time. The smallest and fastest is the “X” memory with a capacity of 64 19-digit words; external word selection and 2 cores per bit make possible an access time of 0.8 μsec and a cycle time of 4 μsec.

“S” Memory (65,536 Words)

The “S” memory (Fig. 1) is a coincident-current magnetic core unit with a storage capacity of 65,536 37-bit words. The bits in the word are read out in parallel with a cycle time of 6.5 μsec and an access time of 2.8 μsec. (Cycle time is the time between successive strobe pulses and access time is the minimum delay between setting the address register and strobing.) The block diagram (Fig. 2, opposite) shows that two 256 position magnetic core switches are used to supply the READ and WRITE current pulses to the X and Y selection lines. The operating characteristics of these switches are such that the contents of the address register are no longer needed after the READ half of the cycle, and the interval between READ and WRITE may be extended several microseconds under computer control to permit the other operations to occur. Two coordinates are used to select a register during READ and three coordinates are used for WRITE. In each case, 2:1 current selection ratio is used. The S memory with 604 tubes and 1406 transistors, is a 37-digit version of the 19-digit TX-0 memory that has been described in the literature.1 The basic operation of this type memory has also been described and will not be repeated here.2

* This work was supported jointly by the Army, Navy, and Air Force under contract with the Mass. Inst. Tech.
† M.I.T. Lincoln Lab., Lexington, Mass.

Fig. 1—“S” memory, 65,536 words, 37 digits.